**ソースコード説明書(SWEET ENGINEER2)**

※ソースコードはフォルダ分けしていませんが、Visual Studioのソリューションエクスプローラ上でフィルタ分けしています。

　※このプロジェクトは元々ゲーム大賞に向けてチームで完成させたものを、引き続き個人で拡張していっているものです。説明書内に記載されていない部分に関しては別の担当者がプログラムしたものです。また、説明書内に『別の担当者がプログラムしたファイルである』と明記している場合もあります。

●最新、または特にアピールしたい部分。

・フィルタ Game\Enemy\Boss内のソースコード

こちらは個人制作に映ってから作成したコードです。これまで本格的に担当してこなかったゲーム部分のプログラミングに挑戦しました。主にボスの挙動を定義したもので、直進や回転などの細かい挙動をステート化して使い回したかったため、ヒエラルキーステートパターンで実装しています。ステートのコードは同フィルタにあるHFSMフィルタの中にあります。

また、ボスが逃げる道筋は拡張性を考慮してUnityで配置し、エディタ拡張機能でヘッダファイルに書き出しています。また、道筋はボックスを組み合わせて定義しており、各ボックスにはボスのステートを決定づける属性を設定できるようにしています。これにより、ボスが逃げる途中で待機したり、攻撃に移ったりするようにしています。

・フィルタ Game\Field\CourceDef.cppおよびCourceDef.h

　プレイヤーが進む道筋やボスの逃走経路を定義しているファイルです。

CCourceDefクラスはプレイヤーの道筋を定義したものです。こちらはチーム制作時点で作成したものです。UnityでCourceという名前のルートを決め、その下にボックスをどんどん子供に設定していく形で配置しています。このクラスは各コースブロック(コースを形成している１つの塊。これを組み合わせて複雑な道筋を表現する)をノード形式で生成しているクラスです。コースブロックはそれぞれ向きを持っており、プレイヤーやカメラはそれを参照して自分の進む方向を決定します。各コースブロックは親ボックスから子ボックスに向けてのベクトルを算出し、それをコースの向きとしています。また、プレイヤーがどのコースブロックにいるかの判定は、スタートブロックのポジションからプレイヤーのポジションへのベクトルを求め、内積を用いてコースの向きベクトルに射影した結果求まる長さで判定しています。

　CBossCourceクラスはボスの道筋を定義したクラスです。CCourceDefと違い、Unityであらかじめコースブロックを定義してから出力しています。また、ボスのコースブロックはコリジョンオブジェクトを生成し、物理エンジンを用いた衝突判定でボスが今いるコースブロックを導き出しています。これは、CCourceDefクラスの方式では２つのコースブロックが重なった場合に予期せぬ挙動をしたためです。プレイヤーのコースはデータで調整してこれを解決したのですが、その調整にとても時間がかかりました。なので、ボスの逃走経路は保守性と拡張性を考慮して新しいクラスを作りました。コースブロックが持つ範囲のコリジョンは、Unityに範囲用のボックスを配置し、それのScaleを用いてコリジョンを生成しています。

・フォルダProject\ChocoBall\Shader内のソースコード

※こちらはフィルタではなくフォルダです。

直下にあるのはこのゲームに使用しているグラフィックテクニックを実装するために私が作成したものです。プログラマブルシェーダーで頂点やピクセルにプログラミングしています。

※以下担当部分と簡単な解説になります。

●フィルタ『Engine』

　ほとんど私が担当した部分です。

ゲームのエンジン部分(どんなゲームであれ、たいてい必要となる共通のファイル)となります。

このフィルタ内ではさらにいくつかのフィルタに区分しています。Engineフィルタの直下にあるファイルはそのどれにも当てはまらない、もしくは複数のグループに該当するファイルです。

・フィルタ『Audio』

BGM、効果音に関するファイルをまとめたものです。ここにあるものは、他のプログラマから頂いたものです。

・フィルタ『Debug』

デバッグ用にあれば便利、もしくは原因不明のバグを生み出しにくくするためのものをまとめたものです。

Assertファイル内の関数を、エンジンの関数の処理の直前などで呼び出すことで、予期せぬNULLアクセスなどを未然に防ぎ、エンジン関数内でNULLアクセスなどが発生したことをプログラマに通知することができます。

PixTagは、PixTagクラス内にある関数を、各描画関数の開始行と終了行で呼び出すことによってタグを打ち込んでくれます。これにより、PixWinを使って1フレームの描画を取得してデバッグする際、どこからどこまでがどの描画関数によるものなのかがよりわかりやすくなります。

・フィルタ『GameObject』

ゲームで使用する、オブジェクトに関するファイルをまとめたものです。

GameObjectクラスは、ゲーム内で使用するオブジェクトに必須となる項目や処理をまとめた基底クラスで、画面に表示するオブジェクトなどにこれを継承させて使用します。

ObjectManagerクラスは、GameObjectクラスを継承したオブジェクトを登録することで、更新、描画、削除などの処理を行ってくれます。このクラスを利用することで、各オブジェクトの更新、描画、削除関数の呼び忘れなどの手間が省けるとともに、ヒューマンエラーの発生確率を減らすことができます。

・フィルタ『Graphics』

オブジェクトの描画に関するファイルをまとめたものです。

オブジェクトを描画するRenderクラスや、描画の際に使用する値を格納するModelクラス、シェーダなどを利用した描画テクニックのクラスなどがあります。また、オブジェクトの位置や向きに影響されず必ず決まった当たり方をするライトのクラス、ActerLightなどもあります。

フィルタ『PostEffect』：

基本描画(３Dオブジェクトの描画)終了後に、輝度などを抽出

したテクスチャを用いてブラーをかけるBloomクラスや、スクリーンに映っているオブジェクトの深度値が書き込まれたテクスチャを利用してブラーをかけるDof(被写界深度)クラスなどがあります。

フィルタ『PreRender』：

丸影などではなく、実際のオブジェクトの形をした、アニメーションに合わせて変形するリアルな影を描画するクラスや、物理ベースレンダリングのフレネルに使用する環境マップを描画するクラスなどがあります。

・フィルタ『Input』

　入力インターフェースに関するファイルをまとめたものです。

　DirectInputやXInputを利用しています。

　キーボードでの入力判定を行うKeyBoardクラスとゲームパッドでの入力判定を行うGamePadクラスがあります。また、それらの基底クラスとなるInterfaceクラスもあります。

　ゲームパッドを使用している場合はそちらの判定が優先され、キーボードでの入力は受け付けないようにしています。また、１フレームごとにゲームパッドの接続状態を判定することで、ゲーム中にゲームパッドが抜かれれば自動的にキーボードでの判定に、ゲームパッドが接続されると自動でゲームパッドでの判定に切り替わるようにしています。

・フィルタ『Particle』

　パーティクルに関する処理をまとめたものです。

粒子の一粒分の処理をまとめたParticleクラスと、パーティクルを発生させるParticleEmitterクラスがあります。

　また、ParticleParameterTableは、使用するテクスチャファイルや重力、パーティクルの初期速度や加速度などを、構造体に格納し、エミッターにそれを渡すことで様々なパーティクルを容易に表現、調整するためのものです。

・フィルタ『Random』

Randよりも精度の高いランダムのクラスです。このフィルタ内のファイルは他のプログラマから譲り受けたものです。

・フィルタ『Timer』

　おもにFPSなどを計測するために用います。このフィルタ内のファイルは他のプログラマから譲り受けたものです。

●フィルタ『Game』

　主に私以外のプログラマが担当した部分です。

そのゲーム独自の仕様、及び遊びの部分を中心としたファイルがまとめられています。

このフィルタ内ではさらにいくつかのフィルタに区分しています。Gameフィルタの直下にあるファイルはそのどれにも当てはまらない、もしくは複数のグループに該当するファイルです。

・フィルタ『Enemy』

　エネミーに関するファイルをまとめたものです。

　EnemyBullet.h及びEnemyBullet.cpp以外は他のプログラマが作成したものです。このEnemyBulletクラスはエネミーが撃つ弾のクラスです。

このフィルタ内のEnemyBase.h、及びEnemyBase.cppは他のプログラマが作成したものですが、これを継承した派生クラスに共通処理が多く見られたため、それらをEnemyBaseクラス内の関数に移植しました。

移植する際、列挙型を利用したステートパターンにリファクタリングしています。

細部が異なる部分は仮想関数を定義し、それを基底クラスで呼び出しています(継承クラスでこの関数をオーバーライドすれば自動でそちらが呼ばれるため、そこに細部違う処理を記載しました)。

・フィルタ『Gimmick』

　ギミックに関するファイルをまとめたものです。

　FallFloor、FireJet、JetGimmick、SmokeJetのヘッダとソースは私が作成したものです。

・フィルタ『HUD』

ゲームシーン(メインシーン)で使用する２D画像に関するファイルをまとめたものです。

Icon、Numberのヘッダとソースは私が作成したものです。

・フィルタ『Field』

コースに沿ったカメラなどの、フィールド、コースに関連するものをまとめたものです。

Skybox以外のヘッダとソースは私が作成したものです。

Fieldクラスの関数内の頂点バッファーにD3DXVECTOR3の値を埋め込んでいる処理は、VSMによって発生した不具合を解消するためのものです。

・フィルタ『Player』

Viblationのソースとヘッダは私が作成したものです。プレイヤーが敵などに衝突したときに、プレイヤーをしびれさせるクラスです。

Playerクラスは他のプログラマが作成したものを、私がリファクタリングと拡張を行ったものです。列挙型を使用したステートパターンを使用するようにしました。また、ジャンプの処理もアニメーションを追加した際、ステートパターンに変更しています。

・フィルタ『Physics』

　物理エンジンに関するファイルをまとめたものです。

　私が作成した部分はありません。

※以下は直下のファイルです。

CourceCamera：

このゲームのカメラは道筋に従って向きを自動で変えます。CourceCameraクラスにはそのロジックを組み込んでいます。

DirectX：

DirectXとは名ばかりのゲームループを記載したファイルです。

GameCamera：

このゲームで使用するカメラの基底クラスです。

カメラのインスタンスの管理を楽にするために、エンジンのObjectManager(ゲームオブジェクトを管理する)クラスに丸投げするため、エンジンのGameObjectクラスを継承しています。カメラの機能自体はエンジンのCameraクラスに委譲しています。

MainScene：

エンジンのGameManagerクラスに登録して使用する、Sceneクラスを継承したクラスです。メインシーンではさらにStageManagerクラスを呼び出しています。

Stage：

StageManagerクラスによってインスタンスが生成されます。ステージが切り替わる際にそのインスタンスは破棄され、また新しいStageインスタンスが生成されます。

ステージの番号によって読み込むステージのXファイルなどを指定します。

StageManager：

Stageインスタンスを生成し、ステージ番号を管理します。また、当初の予定では１ステージクリアすればリザルトシーンに遷移するようになっていたので、一度メインシーンからリザルトシーンに遷移させる処理を呼び出しているのですが、現在リザルトシーンは機能していません。

StageTable：

ステージ１から最終までの情報を設定するファイルです。情報とは主に、BGMファイルの名前、ステージのあたりデータ、エネミーやギミックの配置、ディフューズライトの向きや色などが挙げられます。

ステージのあたりデータや、エネミーとギミックの配置情報などは、Unityで配置した後にツールを使用してヘッダーファイルに位置情報、回転情報、拡縮情報を書き出しています。StageTableでは、そのヘッダを配列内にincludeという形で展開しています。

TestInstancing：

インスタンシング描画を実装するときに作成したテスト用のクラスです。途中で興が乗ってしまい、ボール状のチョコレートの塊が脈打つ(値によっては爆散した後に再集合する)という無駄機能をつけてしまいました。

それを見た担任の先生から「面白い！ ボスの登場とかに使えそう！」との言葉をいただいたのでそのままにしています。なお、未だボス戦にはこの機能は使用されていません。また、このクラスのインスタンスは現在生成すらされていません。